第 3 章

群智能计算

3.1 粒子群优化算法

3.1.1 粒子群优化算法简介

粒子群优化(Particle Swarm Optimization,PSO)算法是进化计算的一个分支,是一种模拟自然界的生物活动的随机搜索算法,又称为粒子群算法、微粒群算法或微粒群优化算法。通常认为 PSO 算法是智能算法的一种,也可以归类为多主体优化系统(Multiagent Optimization System,MAOS)。

PSO 算法模拟了自然界鸟群捕食和鱼群捕食的过程。通过群体中的协作寻找到问题的全局最优解。它在 1995 年由美国学者 Eberhart 和 Kennedy 提出,现在已经广泛应用于各种工程领域的优化问题。

1. 思想来源

PSO 算法思想的起源主要基于生物界现象和社会心理学,生物界现象包括群体行为、群体迁徙、生物觅食等;社会心理学包括群体智慧、个体认知、社会影响等。

PSO 算法是 Kennedy 和 Eberhart 受人工生命研究结果的启发、通过模拟鸟群觅食过程中的迁徙和群聚行为而提出的一种基于群体智能的全局随机搜索算法,自然界中各种生物体均具有一定的群体行为,而人工生命的主要研究领域之一是探索自然界生物的群体行为,从而在计算机上构建其群体模型。自然界中的鸟群和鱼群的群体行为一直是科学家的研究兴趣,生物学家 Craig Reynolds 在 1987 年提出了一个非常有影响的鸟群聚集模型,在他的仿真中,每一个个体遵循:

(1) 避免与邻域个体相冲撞;

(2) 匹配邻域个体的速度;

(3) 飞向鸟群中心,且整个群体飞向目标。

仿真中仅利用上面三条简单的规则,就可以非常接近地模拟出鸟群飞行的现象。1995

年，美国社会心理学家 James Kennedy 和电气工程师 Russell Eberhart 共同提出了粒子群算法，其基本思想是受对鸟类群体行为进行建模与仿真的研究结果的启发。他们的模型和仿真算法主要对 Frank Heppner 的模型进行了修正，使粒子飞向解空间并在最好解处降落。

2. 算法背景

PSO 算法的基本核心是利用群体中的个体对信息的共享从而使整个群体的运动在问题求解空间中产生从无序到有序的演化过程，从而获得问题的最优解。设想这么一个场景：一群鸟进行觅食，而远处有一片玉米地，所有的鸟都不知道玉米地到底在哪里，但是它们知道自己当前的位置距离玉米地有多远。那么找到玉米地的最佳策略，也是最简单有效的策略就是搜寻目前距离玉米地最近的鸟群的周围区域。

在 PSO 算法中，每个优化问题的解都是搜索空间中的一只鸟，称为“粒子”，而问题的最优解就对应于鸟群寻找的“玉米地”。所有的粒子都具有一个位置向量（粒子在解空间的位置）和速度向量（决定下次飞行的方向和速度），并可以根据目标函数计算当前所在位置的适应度值（fitness value），可以将其理解为距离“玉米地”的距离。在每次的迭代中，种群中的粒子除了根据自身的经验（历史位置）进行学习以外，还可以根据种群中最优粒子的经验进行学习，从而确定下一次迭代时需要如何调整和改变飞行的方向和速度。就这样逐步迭代，最终整个种群的粒子就会逐步趋于最优解。

3. 基本原理

鸟群觅食与 PSO 算法在一定程度上存在着类比关系。

（1）鸟群觅食现象：鸟群、觅食空间、飞行速度、所在位置、个体认知与群体协作、找到食物。

（2）PSO 算法：搜索空间的一组有效解、问题的搜索空间、解的速度向量、解的位置向量、速度与位置的更新、找到全局最优解。

图 3.1 和图 3.2 分别描述的是鸟群觅食现象以及 PSO 算法。

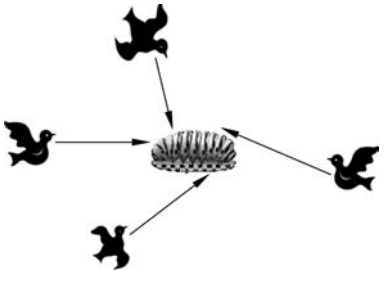


图 3.1　鸟群觅食现象

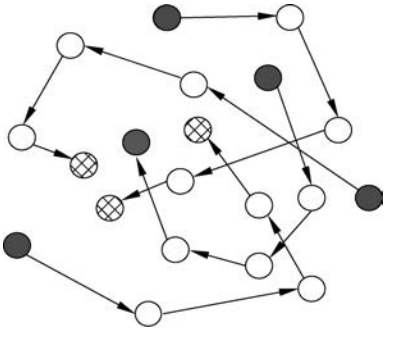


图 3.2　PSO 算法

4. 算法定义

PSO 算法模拟鸟群的捕食行为。一群鸟在随机搜索食物，在这个区域里只有一块食物。所有的鸟都不知道食物在哪里。但是它们知道当前的位置离食物还有多远。那么找到食物的最优策略是什么呢？最简单有效的就是搜寻离食物最近的鸟的周围区域。

根据鸟群的捕食行为模型，PSO 算法得到启示并用于解决优化问题。PSO 算法中，每个优化问题的解都是搜索空间中的一个粒子。所有的粒子都有一个由被优化的函数决定的适应度值，每个粒子还有一个速度决定它们的方向和距离。然后粒子们就追随当前的最优粒子在解空间中搜索。

PSO 算法初始化为一群随机粒子（随机解），通过迭代找到最优解。在每一次迭代中，粒子通过跟踪两个“极值”更新自己。第一个就是粒子本身所找到的最优解，这个解叫作个体极值 pBest，另一个极值是整个种群找到的最优解，这个极值是全局极值 gBest。另外也可以不用整个种群而是用其中一部分最优粒子的邻居，那么在所有邻居中就是局部极值。

应用 PSO 算法解决优化问题的过程中有两个重要的步骤：问题解的编码和适应度函数（fitness function）。PSO 算法的一个优势就是采用实数编码，不需要像遗传算法一样采用二进制编码。

PSO 算法中并没有特别多需要调节的参数，以下列出了常用参数以及经验设置。

（1）粒子数一般取 20～40。对于大部分的问题，10 个粒子已经足够取得好的结果。对于比较难的问题或者特定类别的问题，粒子数可以取到 100 或 200。

（2）粒子的长度是由优化问题决定，就是问题解的长度。

（3）粒子的范围由优化问题决定，每一维可是设定不同的范围。

（4）$V_{\max}$ 表示最大速度，决定粒子在一个循环中最大的移动距离，通常设定为粒子的范围宽度。

（5）学习因子 c_1 和 c_2 一般取值为 2。在其他文献中也有不同的取值，如 c_1 等于 c_2 且范围为 0～4。

（6）中止条件即最大循环数以及最小错误要求，中止条件由具体的问题确定。

（7）全局 PSO 和局部 PSO，前者速度快不过有时会陷入局部最优，后者收敛速度慢一点不过很难陷入局部最优。在实际应用中，可以先用全局 PSO 找到大致的结果，再用局部 PSO 进行搜索。

3.1.2 粒子群优化算法的基本流程

假设粒子群优化算法的基本条件如下：

（1）目标搜索空间为 D 维空间。

（2）粒子群由 N 个粒子组成。

（3）第 i 个粒子在 D 维空间的位置为 $x_i=\{x_i^1,x_i^2,\cdots,x_i^D\}$

(4) 第 i 个粒子的飞翔速度为 $v_i=(v_i^1,v_i^2,\cdots,v_i^D)$

(5) 第 i 个粒子的最优位置 $\mathrm{pBest}_i=(\mathrm{pBest}_i^1,\mathrm{pBest}_i^2,\cdots,\mathrm{pBest}_i^D)$

(6) 全局最优位置 $\mathrm{gBest}=(\mathrm{gBest}^1,\mathrm{gBest}^2,\cdots,\mathrm{gBest}^D)$

则 PSO 算法粒子的个体速度与位置更新公式分别为

$$v_i^d=\omega\times v_i^d+c_1\times r_1^d\times(\mathrm{pBest}_i^d-x_i^d)+c_2\times r_2^d\times(\mathrm{gBest}^d-x_i^d) \tag{3-1}$$

$$x_i^d=x_i^d+v_i^d \tag{3-2}$$

式中，$d=1,2,\cdots,D$；c_1,c_2 为非负常数；$i=1,2,\cdots,m$；r_1,r_2 是介于[0,1]的随机数；$v_i^d\in[-v_{\max},v_{\max}]$由用户设定。

图 3.3 描述的是粒子群算法的概述。速度更新公式由三部分组成。

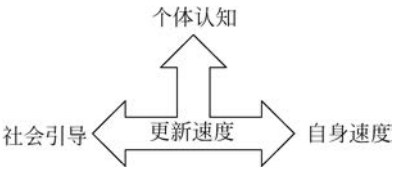


图 3.3　算法概述

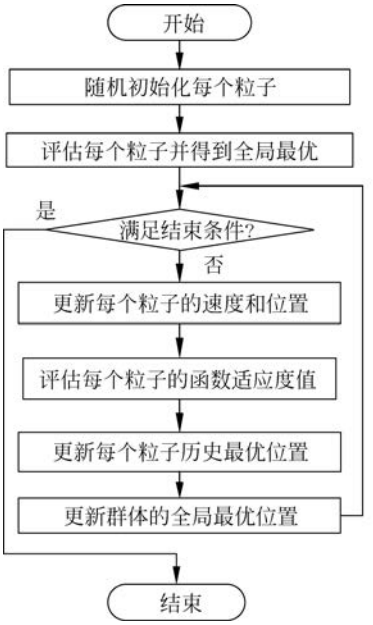
(1) 自身速度：也是惯性或动量部分，反应粒子的运动习惯。

(2) 个体认知：粒子有向自身历史最佳位置逼近的优势。

(3) 社会引导，粒子有向群体或领域历史最佳位置逼近的趋势。

因此式(3-1)中，粒子第 d 步的速度＝上一步自身的速度惯性＋个体认知部分＋社会认知部分，即速度为三部分的和。

粒子群算法的流程和伪代码如图 3.4 所示。




```
//功能：粒子群优化算法伪代码
//说明：本例以求问题最小值为目标
//参数：N为群体规模

procedure PSO
  for each particle i
    Initialize velocity V_i and position X_i for particle i
    Evaluate particle i and set pBest_i = X_i
  end for
  gBest = min {pBest_i}
  while not stop
    for i=1 to N
      Update the velocity and position of particle i
      Evaluate particle i
      if fit (X_i) < fit (pBest_i)
          pBest_i = X_i;
      if fit(pBest_i) < fit (gBest)
          gBest = pBest_i;
    end for
  end while
  print gBest
end procedure
```

图 3.4　算法流程图

人工神经网络(Artificial Neural Network,ANN)是模拟大脑分析过程的简单数学模型,反向转播算法是最流行的神经网络训练算法,但近来也有很多学者利用演化计算(evolutionary computation)技术研究人工神经网络的各个方面。

演化计算一般研究神经网络的三个方面:网络连接权重、网络结构(网络拓扑结构和传递函数)和网络学习算法。目前工作大多数集中在网络连接权重和网络拓扑结构上。在遗传算法中,网络权重和/或拓扑结构一般编码为染色体(Chromosome),适应度函数的选择一般根据研究目的确定,例如在分类问题中,错误分类的比率可以用作适应度值。

【例 3-1】 使用分类问题的基准函数(benchmark function)IRIS 数据集说明 PSO 算法训练神经网络的过程。在数据集中,每组数据包含鸢尾花的 4 种属性:萼片长度、萼片宽度、花瓣长度和花瓣宽度。3 种不同的花各有 50 组数据,总共有 150 组数据或模式。

解:使用 3 层的神经网络进行分类,则网络有 4 个输入和 3 个输出,所以神经网络的输入层有 4 个节点,输出层有 3 个节点,隐层节点的数目可以动态调节,这里假定隐层有 6 个节点。神经网络中的参数都可以进行训练,这里只确定网络权重。粒子就表示神经网络的一组权重,应该是 $4\times6+6\times3=42$ 个参数。权重的范围设定为$[-100,100]$。完成编码以后还需要确定适应度函数。对于分类问题,把所有数据送入神经网络,网络的权重由粒子的参数决定。记录所有的错误分类的数目作为粒子的适应值。利用 PSO 算法训练神经网络可以获得尽可能低的错误分类数目。PSO 算法本身并没有很多的参数需要调整,所以在实验中只需要调整隐层的节点数目和权重的范围就可以取得较好的分类效果。

3.1.3 粒子群算法分类

1. 标准 PSO 算法的变形

在标准 PSO 算法的变形中,主要是对标准 PSO 算法的惯性因子、收敛因子(约束因子)、"认知"部分的学习因子 c_1,"社会"部分的学习因子 c_2 进行变化与调节,希望获得好的效果。

惯性因子的原始版本是保持不变的,后来有人提出随着算法迭代的进行,惯性因子需要逐渐减小的思想。算法开始阶段,大的惯性因子可以使算法不容易陷入局部最优,到算法后期,小的惯性因子可以使收敛速度加快,收敛更加平稳,不至于出现振荡现象。经过测试,动态地减小惯性因子 w,的确可以使算法更加稳定,效果比较好。但是递减惯性因子采用什么样的方法呢?人们首先想到的是线性递减,这种策略的确很好,但是是不是最优的呢?于是有人对递减的策略做了相关研究,研究结果指出:线性函数的递减优于凸函数的递减策略,但是凹函数的递减策略又优于线性的递减,经过测试,实验结果基本符合此结论,但效果不是很明显。

对于收敛因子,经过证明如果收敛因子取 0.729,可以确保算法的收敛,但是不能保证算法收敛到全局最优,经过测试,取收敛因子为 0.729 效果较好。对于学习因子 c_2 和 c_1 也

有人提出：c_1 先大后小，而 c_2 先小后大的思想。因为在算法运行初期，每个粒子要有大的自己的认知部分而又比较小的社会部分，这个与一群人找东西的情形比较接近，因为在找东西的初期，基本依靠自己的知识去寻找，随着积累的经验越来越丰富，于是大家开始逐渐达成共识(社会知识)，这样就开始依靠社会知识寻找东西了。

2007 年，希腊的两位学者提出将收敛速度比较快的全局版本的 PSO 算法与不容易陷入局部最优的局部版本的 PSO 算法相结合的办法，速度更新公式和位置更新公式分别为

$$v = n \times v_{\text{全局版本}} + (1-n) \times v_{\text{局部版本}} \tag{3-3}$$

$$w(k+1) = w(k) + v \tag{3-4}$$

该算法在文献中讨论了系数 n 取各种不同情况的情况，并运行了近 20000 次后分析各种系数的结果。

2. PSO 算法的混合

PSO 算法可以与各种算法相混合，例如与模拟退火算法相混合或与单纯形方法相混合等。但是使用最多的是将 PSO 算法与遗传算法结合，根据遗传算法的 3 种不同算子可以生成 3 种不同的混合算法。

(1) PSO 算法与选择算子的结合，结合思想：在原来的 PSO 算法中选择粒子群群体的最优值作为 p_g，但是相结合的版本是根据所有粒子的适应度的大小给每个粒子赋予一个被选中的概率，然后依据概率对这些粒子进行选择，被选中的粒子作为最优，其他情况保持不变。这样的算法可以在算法运行过程中保持粒子群的多样性，但是致命的缺点是收敛速度缓慢。

(2) PSO 算法与交叉算子的结合，结合思想：在算法运行过程中根据适应度的大小，粒子之间可以两两交叉，比如用一个很简单的公式：

$$w(\text{新}) = n \times w_1 + (1-n) \times w_2 \tag{3-5}$$

w_1 与 w_2 就是这个新粒子的父辈粒子。这种算法可以在算法的运行过程中引入新的粒子，但是算法一旦陷入局部最优，那么粒子群算法将很难摆脱局部最优。

(3) PSO 算法与变异算子的结合，结合思想：测试所有粒子与当前最优的距离，当距离小于一定的数值的时候，可以对部分粒子进行随机初始化，让这些粒子重新寻找最优值。

3. 二进制 PSO 算法

最初的 PSO 算法是从解决连续优化问题发展起来的。Eberhart 等又提出了 PSO 算法的离散二进制版，用来解决工程实际中的组合优化问题。他们在提出的模型中将粒子的每一维及粒子本身的历史最优、全局最优限制为 1 或 0，而速度不做这种限制。用速度更新位置时，设定一个阈值，当速度高于该阈值时，粒子的位置取 1，否则取 0。二进制 PSO 算法与遗传算法在形式上很相似，但实验结果显示，在大多数测试函数中，二进制 PSO 算法比遗传算法速度快，尤其在问题的维数增加时。

4. 协同 PSO 算法

协同 PSO 算法将粒子的 D 维分到 D 个粒子群中，每个粒子群优化一维向量，评价适应度时将这些分量合并为一个完整的向量。例如第 i 个粒子群，除第 i 个分量外，其他 $D-1$ 个分量都设为最优值，不断用第 i 个粒子群中的粒子替换第 i 个分量，直到得到第 i 维的最优值，其他维相同。为将有联系的分量划分在一个群，可将 D 维向量分配到 m 个粒子群优化，则前($D \bmod m$)个粒子群的维数是 D/m 的向上取整。后 $m-(D \bmod m)$个粒子群的维数是 D/m 的向下取整。协同 PSO 算法在某些问题上有更快的收敛速度，但该算法容易被欺骗。

5. 混合策略 PSO 算法

混合策略混合 PSO 算法就是将其他进化算法、传统优化算法或其他技术应用到 PSO 算法中，提高粒子多样性，增强粒子的全局探索能力或者提高局部开发能力，增强收敛速度与精度。混合策略通常有两种：一是利用其他优化技术自适应调整收缩因子/惯性值、加速常数等；二是将 PSO 算法与其他进化算法的操作算子或技术结合，例如将蚂蚁算法与 PSO 算法混合用于求解离散优化问题。

Robinson 和 Juang 将遗传算法与 PSO 算法结合分别用于天线优化设计和递归神经网络设计，并将种群动态划分成多个子种群，再对不同的子种群利用 PSO 算法、遗传算法或爬山法进行独立进化。Naka 等将遗传算法中的选择操作引入 PSO 算法中，按一定选择率复制较优个体。Angeline 则将锦标赛选择引入 PSO 算法，根据个体当前位置的适应度，将每个个体与其他若干个个体相比较，然后依据比较结果对整个群体进行排序，用粒子群中最好一半的当前位置和速度替换最差一半的位置和速度，同时保留每个个体所记忆的个体最好位置。EDib 等对粒子位置和速度进行交叉操作。Higashi 将高斯变异引入 PSO 算法中。Miranda 等则使用了变异、选择和繁殖多种操作同时自适应确定速度更新公式中的邻域最佳位置以及惯性权重和加速常数。Zhang 等利用差分进化操作选择速度更新公式中的粒子最佳位置。而 Kannan 等则利用差分进化优化 PSO 算法的惯性权重和加速常数。

常见混合策略 PSO 算法主要包括以下几种。

(1) 高斯 PSO(Gaussian PSO，GPSO)算法。由于传统 PSO 算法往往是在全局和局部最佳位置的中间进行搜索，搜索能力和收敛性能严重依赖加速常数和惯性权重的设置，为了克服该不足，Secrest 等将高斯函数引入 PSO 算法中，用于引导粒子的运动。GPSO 算法不再需要惯性权重，而加速常数由服从高斯分布的随机数产生。

(2) 拉伸 PSO(Stretching PSO，SPSO)算法将所谓的拉伸技术(stretching technique)以及偏转和排斥技术应用到 PSO 算法中，对目标函数进行变换，限制粒子向已经发现的局部最小解运动，从而使粒子有更多的机会找到全局最优解。

(3) 混沌 PSO(Chaos PSO，CPSO)算法。混沌是自然界一种看似杂乱、其实暗含内在规律性的常见非线性现象，具有随机性、遍历性和规律性等特点。以粒子群的历史最佳位置

为基础产生混沌序列，并将此序列中的最优位置随机替代粒子群中的某个粒子的位置，就形成了 CPSO 算法。还有利用惯性权重自适应于目标函数值的自适应 PSO 算法进行全局搜索，利用混沌局部搜索对最佳位置进行局部搜索的混沌 PSO 算法。

(4) 免疫 PSO 算法。生物免疫系统是一个高度鲁棒性、分布性、自适应性并具有强大识别能力、学习和记忆能力的非线性系统。已有文献将免疫系统的免疫信息处理机制(抗体多样性、免疫记忆、免疫自我调节等)引入到 PSO 算法中，分别提出了基于疫苗接种的免疫 PSO 算法和基于免疫记忆的免疫 PSO 算法。

(5) 量子 PSO 算法优化将量子个体应用于离散 PSO 算法，并基于量子行为更新粒子位置。

3.1.4 粒子群优化算法的改进研究

PSO 算法的研究热点与方向包括算法理论研究、算法参数研究、拓扑结构研究、混合算法研究、算法应用研究等。

1. 理论研究的改进

Clerc 和 Kennedy 在 2002 年设计了一个称为压缩因子的参数。在使用了此参数之后，PSO 算法能够更快地收敛。Trelea 在 2003 年指出 PSO 算法最终稳定地收敛于空间中的某一个点，但不能保证是全局最优点。Kadirkamanathan 等在动态环境中对 PSO 算法的行为进行研究，由静态分析深入到动态分析。Van den Bergh 等对 PSO 算法的飞行轨迹进行了跟踪，并深入到动态的系统分析和收敛性研究。

2. 拓扑结构的改进

PSO 算法拓扑结构的改进包括以下几种。

(1) 静态拓扑结构，分为全局版本(如星形结构)和局部版本(如环形结构、齿形结构、冯·诺依曼结构)等，图 3.5 给出了 4 种典型的拓扑结构。

(2) 动态拓扑结构，包括 Suganthan 在 1999 年提出的逐步增长法、Hu 和 Eberhart 在 2002 年提出的最小距离法、Liang 和 Suganthan 在 2005 年提出的重新组合法以及 Kennedy 等在 2006 年提出的随机选择法等。

(3) 其他拓扑结构，包括 Kennedy 提出的社会趋同法、Mendes 等提出的 Fully Informed、Liang 等提出的广泛学习策略等。

对于目前使用较多的静态拓扑结构，全局版本 PSO(Global PSO，GPSO)算法和局部版本 PSO(Local PSO，LPSO)算法在收敛特点上有如下区别。

(1) GPSO 算法具有很高的连接度，往往具有比 LPSO 算法更快的收敛速度。但是，快速的收敛也让 GPSO 算法付出了多样性迅速降低的代价。

(2) LPSO 算法由于具有更好的多样性，因此一般不容易落入局部最优，在处理多峰问题上具有更好的性能。

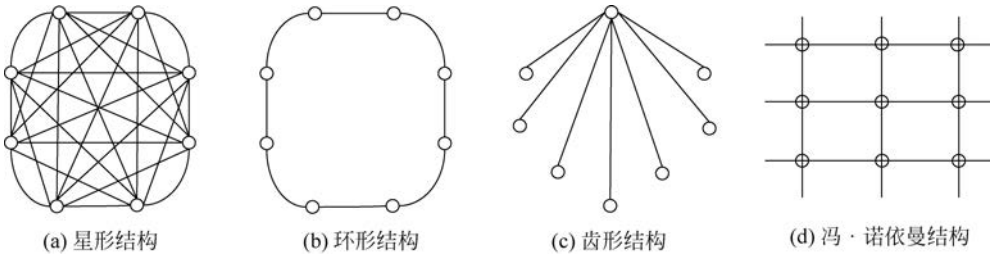


图 3.5　4 种典型的拓扑结构示意图

在使用静态拓扑结构解决具体问题的时候，可以遵循以下一些规律。

(1) 邻域较小的拓扑结构在处理复杂的、多峰值的问题上具有优势，例如环形结构的 LPSO 算法。

(2) 随着邻域的扩大，算法的收敛速度将会加快，这对简单的、单峰值的问题非常有利，例如 GPSO 算法在这些问题上就表现很好。

3. 混合算法改进

(1) 混合进化算子的改进：选择算子、交叉算子、变异算子、进化规划、进化策略。

(2) 混合其他搜索算法的改进：结合模拟退火算法、人工免疫算法、差分进化算法或局部搜索算法等。

(3) 混合其他技术的改进：包括单纯形技术、函数延伸技术、混沌技术、量子技术、协同技术、小生境技术、物种形成技术。

(4) 二进制编码：Kennedy 和 Eberhart 在 1997 年对 PSO 算法进行了离散化，形成了二进制编码 PSO 算法，并且在对 De Jong 提出的 5 个标准测试函数的测试中取得较好的效果。

(5) 整数编码：Salman 等将粒子的位置变量四舍五入为最接近的合法的离散值。Yoshida 等将连续的值域划分为多个区间，每个区间赋予一个相应的离散值

(6) 其他形式：Schoofs 和 Naudts 重新定义了 PSO 算法的“加减乘”方法，并应用于约束可满足问题(Constraint Satisfaction Problem，CSP)中。Hu 等将速度定义为位置变量相互交换的概率，从而将 PSO 算法离散化并用于解决 n-皇后(n-queen)问题。Clerc 为 PSO 算法定义了合适的“加减乘”法实现离散化，并应用于解决旅行商问题。Chen 等基于集合论的技术，重新定义了 PSO 算法的速度和位置更新公式实现了离散化。

3.1.5　粒子群优化算法的参数设置

1. 种群规模 N

种群规模影响着算法的搜索能力和计算量，PSO 算法对种群规模要求不高，一般取 20～40 就可以达到很好的求解效果。不过对于比较难的问题或者特定类别的问题，粒子数

可以取 100 或 200。粒子的长度 D 由优化问题本身决定，就是问题解的长度。粒子的范围 R 由优化问题本身决定，每一维可以设定不同的范围。

2. 最大速度 V_{max}

最大速度决定粒子每一次的最大移动距离，制约着算法的探索和开发能力。V_{max} 的每一维 V_{max}^{d} 一般可以取相应维搜索空间的 10%～20%，甚至 100%。也有研究使用将 V_{max} 按照进化代数从大到小递减的设置方案。

3. 惯性权重 ω

惯性权重控制前一速度对当前速度的影响，用于平衡算法的探索和开发能力一般设置为从 0.9 线性递减到 0.4，也有非线性递减的设置方案可以采用模糊控制的方式设定，或者在[0.5,1.0]之间随机取值，ω 设为 0.729 的同时将 c_1 和 c_2 设为 1.49445，有利于算法的收敛。

4. 压缩因子 χ

压缩因子可以限制粒子的飞行速度，保证算法的有效收敛。Clerc 等通过数学计算得到 χ 取值为 0.729，同时 c_1 和 c_2 设为 2.05。

5. 加速系数 c_1 和 c_2

加速系数 c_1 和 c_1 代表了粒子向自身极值 pBest 和全局极值 gBest 推进的加速权重，c_1 为粒子的个体学习因子，c_2 为粒子的社会学习因子。c_1 和 c_2 通常都等于 2.0，代表着对两个引导方向的同等重视，也存在一些 c_1 和 c_2 不相等的设置，但其范围一般都在 0～4。研究 c_1 和 c_2 的自适应调整方案对算法性能的增强有重要意义。

6. 终止条件

终止条件决定算法运行的结束，由具体的应用和问题本身确定将最大循环数设定为 500、1000、5000 或者最大的函数评估次数，也可以使用算法求解得到一个可接受的解作为终止条件或者是当算法在很长一段迭代中没有得到任何改善，则可以终止算法。

7. 全局和局部 PSO

全局和局部 PSO 决定算法如何选择两种版本的粒子群优化算法——GPSO 和 LPSO，GPSO 速度快，不过有时会陷入局部最优；LPSO 收敛速度慢一点，不过不容易陷入局部最优。在实际应用中，可以根据具体问题选择具体的算法版本。

8. 同步和异步更新

两种更新方式的区别在于对全局 gBest 或者局部 pBest 的更新方式。在同步更新方式中，在每一代中，当所有粒子都采用当前的 gBest 进行速度和位置的更新，之后才对粒子进行评估，更新各自的 pBest，再选择最好的 pBest 作为新的 gBest。

在异步更新方式中，在每一代中，粒子采用当前的 gBest 进行速度和位置的更新，然后马上评估，更新自己的 pBest，而且如果其 pBest 要优于当前的 gBest，则立刻更新 gBest，迅

速将更好的 gBest 用于后面的粒子的更新过程中。

一般而言,异步更新的 PSO 算法具有高效的信息传播能力和更快的收敛速度。

3.1.6 粒子群优化算法与遗传算法的比较

PSO 算法与遗传算法的流程一般都遵循以下步骤。

(1) 种群随机初始化。

(2) 对种群内的每一个个体计算适应度值。适应度值与最优解的距离直接相关。

(3) 种群根据适应度值进行复制。

(4) 如果终止条件满足的话,就停止,否则转步骤(2)。

从以上步骤可以看到 PSO 算法和遗传算法有很多共同之处。两者都可以随机初始化种群,而且都使用适应度值对系统进行评价,而且都根据适应度值进行随机搜索。两个系统都不能保证一定找到最优解。但是,PSO 算法没有交叉和变异等遗传操作,而是根据自己的速度决定搜索。同时,PSO 算法的粒子还有一个重要的特点,就是有记忆。

与遗传算法相比,PSO 的信息共享机制是不同的。在遗传算法中,染色体(chromosomes)互相共享信息,所以整个种群的移动是比较均匀地向最优区域移动。在 PSO 算法中,只有 gBest信息给其他的粒子,这是单向的信息流动,整个搜索更新过程是跟随当前最优解的过程。与遗传算法比较,在大多数的情况下,所有的粒子可能更快地收敛于最优解。

演化计算的优势是可以处理一些传统方法不能处理的问题,例如不可导的节点传递函数或者没有梯度信息等。但是也可能出现在某些问题上性能并不是特别好,同时遗传算子的选择也比较麻烦。

目前,已经有一些利用 PSO 算法代替反向传播算法训练神经网络的论文。研究表明 PSO 算法是一种很有潜力的神经网络算法。

3.1.7 粒子群优化算法的相关应用与 MATLAB 算例

1. PSO 算法应用

PSO 算法用粒子模拟鸟类个体,每个粒子可视为 N 维搜索空间中的一个搜索个体,粒子的当前位置即为对应优化问题的一个候选解,粒子的飞行过程即为该个体的搜索过程。粒子的飞行速度可根据粒子历史最优位置和种群历史最优位置进行动态调整。粒子仅有两个属性:速度和位置,速度代表移动的快慢,位置代表移动的方向。每个粒子单独搜寻的最优解叫作个体极值,粒子群中最优的个体极值作为当前全局最优解。算法不断迭代,更新速度和位置,最终得到满足终止条件的最优解。

PSO 算法的迭代流程如图 3.6 所示,具体流程具体如下。

(1) 初始化。首先设置最大迭代次数、目标函数的自变量个数、粒子的最大度、位置信息为整个搜索空间,在速度区间和搜索空间上随机初始化速度和位置,设置粒子群规模为

M，为每个粒子随机初始化一个飞翔速度。

（2）个体极值与全局最优解。定义适应度函数，个体极值为每个粒子找到的最优解，从这些最优解找到一个全局值，称为本次全局最优解，与历史全局最优比较并进行更新。

（3）更新速度和位置为

$$V_i^d = \omega c_1 \text{random}(0,1)(\text{pEest}_i^d - X_i^d) + c_2 \text{random}(0,1)(\text{pBest}^d - X_i^d) \quad (3\text{-}6)$$

其中，ω 为惯性权重，其值为非负，较大时全局寻优能力强而局部寻优能力弱，较小时全局寻优能力弱而局部寻优能力强。通过调整 w 的大小，可以对全局寻优性能和局部寻优性能进行调整。c_1 和 c_2 为加速系数。Suganthan 的实验表明：c_1 和 c_2 为常数时可以得到较好的解，通常设置 $c_1=c_2=2$，但不一定必须等于 2，一般 $c_1=c_2\in[0,4]$。Random(0,1)表示区间[0,1]上的随机数，pBest_i^d 表示第 i 个变量的个体极值的第 d 维，pBest^d 表示全局最优解的第 d 维。

（4）终止条件：达到设定迭代次数；代数之间的差值满足最小界限。

图 3.6 描述的是 PSO 算法的迭代流程。

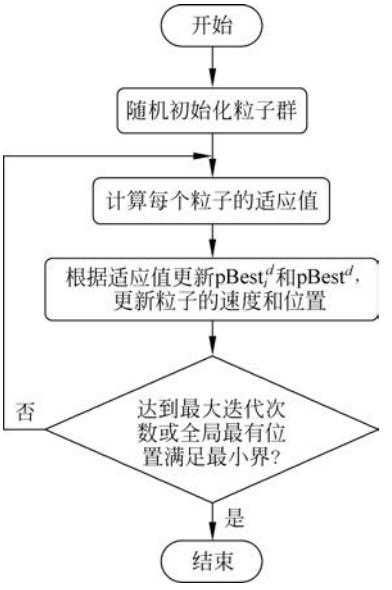


图 3.6　PSO 算法的迭代流程

PSO 算法的一些优点如下。

（1）是一类不确定算法。不确定性体现了自然界生物的生物机制，并且在求解某些特定问题方面优于确定性算法。

（2）是一类概率型的全局优化算法。非确定算法的优点在于算法能有更多机会求解全局最优解。

（3）不依赖优化本身的严格数学性质。

（4）是一种基于多个智能体的仿生优化算法。PSO 算法中的各个智能体之间通过相互协作更好地适应环境，表现出与环境交互的能力。

（5）具有本质并行性。包括内在并行性和内含并行性。

（6）具有突出性。PSO 算法总目标的完成是在多个智能体个体行为的运动过程中突现出来的。

（7）具有自组织和进化性以及记忆功能，所有粒子都保存优解的相关知识。

（8）具有稳健性。稳健性是指在不同条件和环境下算法的实用性和有效性，但是现在 PSO 算法的数学理论基础还不够牢固，算法的收敛性还需要讨论。

2. PSO 算法的 MATLAB 算例

【例 3-2】 求解函数 $f(x)=x\sin x\cos 2x - x\sin 3x$ 的最优解。

解：首先初始化种群。已知位置限制[0,20]，由于一维问题较为简单，因此可以取初始种群 N 为 50，迭代次数为 100，空间维数 $d=1$。位置和速度的初始化即在位置和速度限制

内随机生成一个 $N\times d$ 的矩阵，对于此函数，位置初始化就是在 0～20 内随机生成一个 50×1 的数据矩阵，此处的位置约束也可以理解为位置限制，而速度限制是保证粒子步长不超限制，一般设置速度限制为[－1，1]。粒子群的另一个特点就是记录每个个体的历史最优和种群的历史最优，因此二者对应的最优位置和最优值也需要初始化。其中每个个体的历史最优位置可以先初始化为当前位置而种群的历史最优位置则可初始化为原点。对于最优值，如果求最大值则初始化为负无穷，相反则初始化为正无穷，每次搜寻都需要将当前的适应度值和最优解同历史的记录值进行对比，如果超过历史最优值，则更新个体和种群的历史最优位置。

速度和位置更新是粒子群算法的核心，其原理表达式和更新方式如下：

$$\begin{cases} V_i^d = w \cdot V_i^d + c_1 r_1 (\text{pBest}_i^d - x_i^d) + c_2 r_2 (\text{pBest}^d - x_i^d) \\ x_i^d = x_i^d + v_i^d \end{cases} \tag{3-7}$$

每次更新完速度和位置都需要考虑速度和位置的限制，需要将其限制在规定范围内，此处仅举出一个常规方法，即将超约束的数据约束到边界。图 3.7～图 3.9 分别为粒子的初始状态、最终状态和收敛过程图，由图可知算法已成功找出了最优解，其最优解为 18.3014，而其最大值为 32.1462。

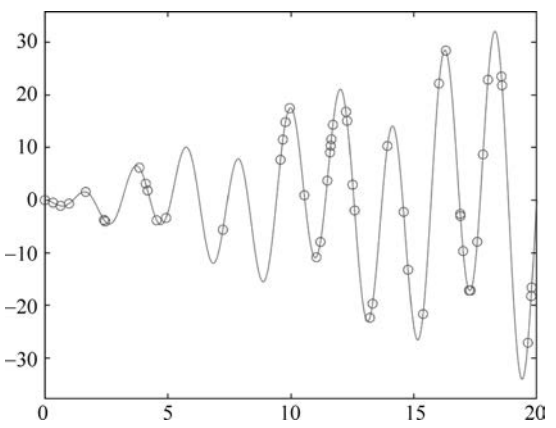


图 3.7　初始粒子状态图

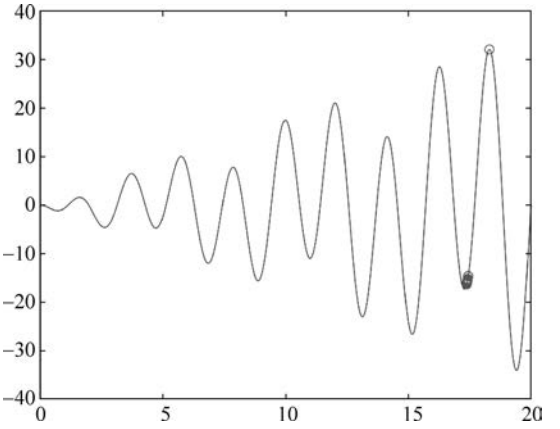


图 3.8　粒子最终状态图

3. 结合 PSO 算法的神经元方法

一般地，可以先使用 PSO 算法在全局范围内进行大致搜索，得到一个初始解，再使用反向传播算法进行更仔细的搜索。

【例 3-3】 对函数 $2.1\times(1-x+2x^2)\mathrm{e}^{\left(-\frac{x^2}{2}\right)}+\sin x+x$，$x\in[-5,5]$进行采样，得到 30 组训练数据，拟合该神经网络。

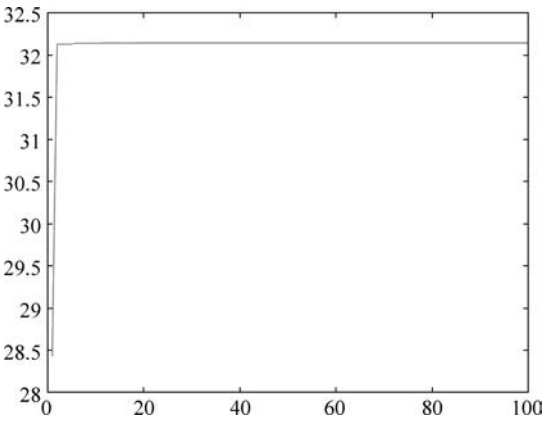


图 3.9　粒子收敛过程

解：该神经网络结构为 1-7-1 结构，包括 1 个输入神经元、7 个中间神经元和 1 个输出神经元。图 3.10 所示为粒子的训练过程，拟合的第一步先抽取 30 组数据，包括输入和输出；第二步运行 PSO 算法，进行随机搜索，选择一个最优的解，该解的维数为 22 维；第三步在 PSO 算法解的基础上使用反向传播算法进行细化搜索。

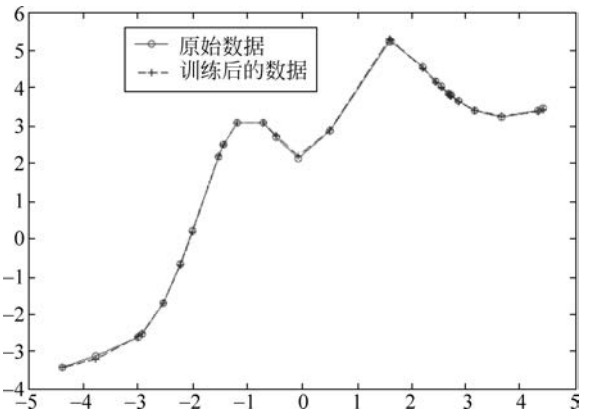


图 3.10　粒子训练过程

3.2　蚁群算法

3.2.1　蚁群算法的基本原理

蚁群算法(Ant Colony Algorithm，ACA)是近几年提出的一种新型的模拟进化算法。意大利学者 Pofigo 基于对自然界中真实蚁群集体行为的研究成果首先提出了 ACA。

以 ACA 为代表的群智能已成为分布式人工智能研究的一个热点，许多源于蜂群和蚁群模型设计的算法越来越多地被应用于企业的运转模式的研究。当前对 ACA 的研究，不

仅有算法意义上的研究，还有从仿真模型角度的研究，并且不断有学者提出对蚁群算法的改进方案。从当前可以查阅的文献情况来看研究和应用蚁群算法的学者主要集中在比利时、意大利、英国、法国、德国等欧洲国家，日本和美国也已经启动对蚁群算法的研究，国内于1998年末开始有少量公开报道和研究成果。

群智能还被应用于工厂生产计划的制定和运输部门的后勤管理。美国太平洋西南航空公司采用一种直接源于蚂蚁行为研究成果的运输管理软件，每年至少节约了1000万美元的费用开支。英国联合利华公司已率先利用群智能技术改善其一家牙膏厂的运转情况。美国通用汽车公司、法国液气公司、荷兰公路交通部和美国一些移民事务机构也都采用这种技术改善其运转的机能。英国电信公司和美国世界通信公司以电子蚂蚁为基础，对新的电信网络管理方法进行了试验。

ACA最初用于解决TSP问题，经过多年发展已经陆续延伸到其他领域中，如图着色问题、大规模集成电路设计、通信网络中的路由问题、负载平衡问题以及车辆调度问题等。蚁群算法在若干领域已获得成功的应用，其中最成功的应用是组合优化问题。

如图3.11和图3.12所示，自然界蚂蚁群体在寻找食物的过程中，通过一种被称为信息素(pheromone)的物质实现相互的间接通信，从而能够合作发现从蚁穴到食物源的最短路径。通过对这种群体智能行为的抽象建模，研究者提出了蚁群优化(Ant Colony Optimization，ACO)算法，为最优化问题尤其是组合优化问题的求解提供了强有力的手段。

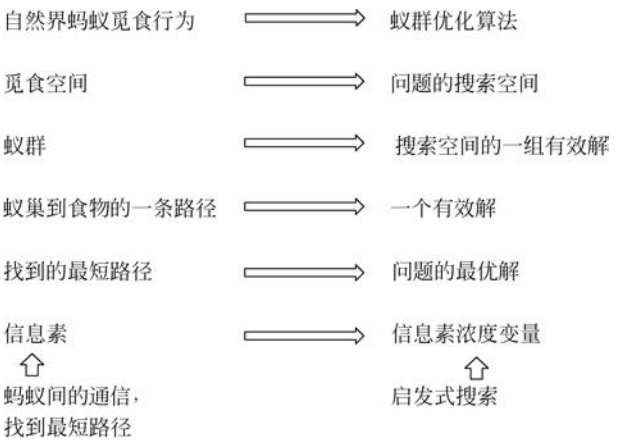


图3.11 描述了蚁群信息搜索的流程图

蚂蚁在寻找食物的过程中往往是随机选择路径的，但它们能感知当前地面上的信息素浓度，并倾向于往信息素浓度高的方向行进。信息素由蚂蚁自身释放，是实现蚁群内间接通信的物质。由于较短路径上蚂蚁的往返时间比较短，单位时间内经过该路径的蚂蚁多，所以信息素的积累速度比长路径快。因此，当后续蚂蚁在路口时，就能感知先前蚂蚁留下的信息，并倾向于选择一条较短的路径前行。这种正反馈机制使得越来越多的蚂蚁在巢穴与食

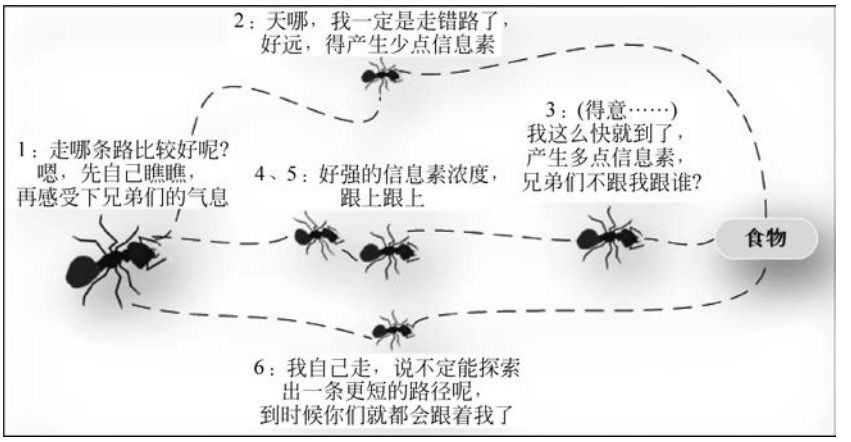


图 3.12　蚁群信息搜索图

物之间的最短路径上行进。由于其他路径上的信息素会随着时间蒸发，最终所有的蚂蚁都在最优路径上行进。

3.2.2　蚁群算法的算法流程

蚂蚁系统(Ant System，AS)是最基本的 ACO 算法，是以 TSP 作为应用实例提出的。

ACO 算法包含两个基本要素。

(1) 路径构建：每只蚂蚁都随机选择一个城市作为其出发城市，并维护一个路径记忆向量，用来存放该蚂蚁依次经过的城市。蚂蚁在构建路径的每一步中，按照一个随机比例规则选择下一个要到达的城市。

(2) 信息素更新：当所有蚂蚁构建完路径后，算法将会对所有的路径进行全局信息素的更新。注意，所描述的是 AS 的 ant-cycle 版本，更新是在全部蚂蚁均完成了路径构造后才进行的，信息素的浓度变化与蚂蚁在这一轮中构建的路径长度相关。

1. 路径构建

对于每只蚂蚁 k，路径记忆向量 $\boldsymbol{R}^k$ 按照访问顺序记录了所有蚂蚁 k 已经经过的城市序号。设蚂蚁 k 当前所在城市为 i，则其选择城市 j 作为下一个访问对象的概率为

$$p_k(i,j)=\begin{cases}\dfrac{[\tau(i,j)]^{\alpha}[\eta(i,j)]^{\beta}}{\sum\limits_{u\in J_k(i)}[\tau(i,u)]^{\alpha}[\eta(i,u)]^{\beta}}, & j\in J_k(i)\\ 0, & \text{其他}\end{cases} \tag{3-8}$$

其中，$J_k(i)$表示从城市 i 可以直接到达的且又不在蚂蚁访问过的城市序列 $\boldsymbol{R}^k$ 中的城市集合。$\eta(i,j)$是一个启发式信息，通常由 $\eta(i,j)=1/d_{ij}$ 直接计算。$\tau(i,j)$表示边(i,j)上的信息素量。α、β 为两常数，分别是信息素和能见度的加权值。

2. 信息素更新

(1) 在算法初始化时,问题空间中所有的边上的信息素都被初始化为 t_0。

(2) 算法迭代每一轮,问题空间中的所有路径上的信息素都会发生蒸发,我们为所有边上的信息素乘上一个小于 1 的常数。信息素蒸发是自然界本身固有的特征,在算法中能够帮助避免信息素的无限积累,使得算法可以快速丢弃之前构建过的较差的路径。

(3) 蚂蚁根据自己构建的路径长度在它们本轮经过的边上释放信息素。蚂蚁构建的路径越短、释放的信息素就越多。一条边被蚂蚁爬过的次数越多、它所获得的信息素也越多。

(4) 迭代步骤(2),直至算法终止。

$$\tau(i,j)=(1-\rho)\cdot\tau(i,j)+\sum_{k=1}^{m}\Delta\tau_k(i,j),$$

$$\Delta\tau_k(i,j)=\begin{cases}(C_k)^{-1}, & (i,j)\in \boldsymbol{R}^k\\ 0, & \text{其他}\end{cases} \tag{3-9}$$

其中,m 是蚂蚁个数;ρ 是信息素的蒸发率,规定 $0<\rho\leqslant 1$。$\Delta\tau_k(i,j)$是第 k 只蚂蚁在它经过的边上释放的信息素量,它等于蚂蚁 k 本轮构建路径长度的倒数。C_k 表示路径长度,它是 $\boldsymbol{R}^k$ 中所有边的长度和。

3. 算法流程实例

【例 3-4】 给出用蚁群算法求解四城市 TSP 问题的执行步骤,4 个城市 A、B、C、D 之间的距离矩阵为

$$\boldsymbol{W}=\boldsymbol{d}_{ij}=\begin{bmatrix}\infty & 3 & 1 & 2\\ 3 & \infty & 5 & 4\\ 1 & 5 & \infty & 2\\ 2 & 4 & 2 & \infty\end{bmatrix}$$

假设蚂蚁种群的规模 $m=3$,参数 $\alpha=1,\beta=2,\rho=0.5$。

解:

步骤 1:初始化。首先使用贪心算法得到路径(ACDBA),则 $C^{mn}=f(\text{ACDBA})=1+2+4+3=10$。求得 $\tau_0=m/C^{mn}=3/10=0.3$。初始化所有边上的信息素 $\tau_{ij}=\tau_0$。

步骤 2.1:为每只蚂蚁随机选择出发城市,假设蚂蚁 1 选择城市 A,蚂蚁 2 选择城市 B,蚂蚁 3 选择城市 D。

步骤 2.2:为每只蚂蚁选择下一个城市。仅以蚂蚁 1 为例,当前城市=A,可访问城市集.合 $J_1(i)=\{\text{B,C,D}\}$。计算蚂蚁 1 选择 B,C,D 作为下一访问城市的概率为

$$\text{A}\Rightarrow\begin{cases}\text{B}:\ \tau_{\text{AB}}^{\alpha}\times\eta_{\text{AB}}^{\beta}=0.3^1\times(1/3)^2=0.033\\ \text{C}:\ \tau_{\text{AC}}^{\alpha}\times\eta_{\text{AC}}^{\beta}=0.3^1\times(1/1)^2=0.3\\ \text{D}:\ \tau_{\text{AD}}^{\alpha}\times\eta_{\text{AD}}^{\beta}=0.3^1\times(1/2)^2=0.075\end{cases}$$

$$p(\mathrm{B})=0.033/(0.033+0.3+0.075)=0.081$$
$$p(\mathrm{C})=0.3/(0.033+0.3+0.075)=0.74$$
$$p(\mathrm{D})=0.075/(0.033+0.3+0.075)=0.18$$

用轮盘选择法则选择下城市。假设产生的随机数 $q=\mathrm{random}(0,1)=0.05$，则蚂蚁 1 将会选择城市 B。

用同样的方法为蚂蚁 2 和 3 选择下一个访问城市，假设蚂蚁 2 选择城市 D，蚂蚁 3 选择城市 A。

步骤 2.3：当前蚂蚁 1 所在城市为 B，路径记忆向量 $\boldsymbol{R}^1=(\mathrm{AB})$，可访问城市集合 $J_1(i)=\{\mathrm{C},\mathrm{D}\}$。计算蚂蚁 1 选择 C、D 作为下一城市的概率为

$$\mathrm{B}\Rightarrow\begin{cases}\mathrm{C}:\ \tau_{\mathrm{BC}}^{\alpha}\times\eta_{\mathrm{BC}}^{\beta}=0.3^1\times(1/5)^2=0.012\\ \mathrm{D}:\ \tau_{\mathrm{BD}}^{\alpha}\times\eta_{\mathrm{BD}}^{\beta}=0.3^1\times(1/4)^2=0.019\end{cases}$$
$$p(\mathrm{C})=0.012/(0.012+0.019)=0.39$$
$$p(\mathrm{D})=0.019/(0.012+0.019)=0.61$$

用轮盘选择法则选择下城市。假设产生的随机数 $q=\mathrm{random}(0,1)=0.67$，则蚂蚁 1 将会选择城市 D。用同样的方法为蚂蚁 2 和 3 选择下一访问城市，假设蚂蚁 2 选择城市 C，蚂蚁 3 选择城市 C。

步骤 2.4：实际上此时路径已经构造完毕，蚂蚁 1 构建的路径为(ABDCA)。蚂蚁 2 构建的路径为(BDCAB)。蚂蚁 3 构建的路径为(DACBD)。

步骤 3：信息素更新。每只蚂蚁构建的路径长度为

$$C_1=3+4+2+1=10,\quad C_2=4+2+1+3=10,\quad C_3=2+1+5+4=12$$

更新每条边上的信息素：

$$\tau_{\mathrm{AB}}=(1-\rho)\times\tau_{\mathrm{AB}}+\sum_{k=1}^{3}\Delta\tau_{\mathrm{AB}}^{k}=0.5\times0.3+(1/10+1/10)=0.35$$

$$\tau_{\mathrm{AC}}=(1-\rho)\times\tau_{\mathrm{AC}}+\sum_{k=1}^{3}\Delta\tau_{\mathrm{AC}}^{k}=0.5\times0.3+(1/12)=0.16$$

……

根据式(3-9)依次计算出问题空间内所有边更新后的信息素量。

步骤 4：如果满足结束条件，则输出全局最优结果并结束程序，否则，转向步骤 2.1 继续执行。

3.2.3　蚁群算法的发展

1. 蚁群算法的参数设置

蚁群算法的参数设置对蚁群算法的性能有着重要的影响。Solion 分析了信息素相关参

数对“勘探”和“开采”的影响，提出了在蚁群算法运行之前加一个预处理阶段，这个阶段先不使用信息素找到一定数量的路径（即回路），再从中选择部分路径在算法开始前初始化信息素，获得了较好的效果。不同的参数组合影响着蚁群算法的性能，Pia 以及 Gaertner 等将遗传算法与蚁群算法相结合，应用遗传算法优化蚁群算法的参数，获得了较好的效果。Meyerl 研究了蚁群算法中参数对“勘探”行为，即保持解的多样性的影响，指出 β 不仅对协调“勘探”和“开采”行为有决定作用，而且对系统的鲁棒性也有重要影响关于蚁群算法参数优化的研究相对较少，然而蚁群算法的参数设置关系到算法最终的性能，因此参数的设置原则值得更深入的研究。

2. 蚁群算法的改进

为了提高蚁群算法的性能，获得更快的收敛速度和求解质量，很多研究工作围绕蚁群算法的改进展开。主要包括如下的改进，Gambardella 等将蚂蚁系统和增强学习中的 Q-学习算法融合在一起提出 AntQ 算法，Dorigo 等采用精英策略（elitist strategy）对蚂蚁系统信息素更新机制进行改进，即增强在每次迭代中找到最优路径的蚂蚁的重要性，对其找到的路径增加额外的信息素，这种策略改善了蚂蚁系统求解大规模问题的能力。Taillard 等提出了快速蚂蚁系统（Fast Ant System，FAS）的概念，FAS 为了避免算法收敛于局部最优而引入了信息素重置机制。

Stutzle 提出了最大-最小蚂蚁系统（MAX-MIN Ant System，MMAS）的概念，它和 3.2.2 节介绍的蚁群系统类似，但 MMAS 事先限定路径上信息素的变化范围为 $[\tau_{\min}, \tau_{\max}]$，$\tau_{\min}$ 和 $\tau_{\max}$ 为预先设定参数，这样可以有效避免搜索陷入停滞。最初的蚁群算法适用于解决组合优化问题，现在也有学者尝试改进蚁群算法求解连续优化问题。Pourtakdoust 等提出了只在信息素指导下进行寻优的求解连续优化问题的蚁群算法。Dec 等提出了连续交互式蚁群（Continuous Interacting Ant Colony，CIAC）算法求解多目标连续函数优化问题。改进蚁群算法求解连续优化问题的研究相对较少，这是一个很有潜力的研究方向。

3. 蚁群算法收敛性的证明

T. Stutztle 等已经证明了 MMAS 的算法收敛性，W. Gutjahr 证明了 GBAS（Graph-Based Ant System）的蚁群能以任意接近的概率收敛到给定的最优解。然而目前 MMAS 的收敛性证明并没有给出收敛速度的估计，而 GBAS 的执行比蚁群有更多的限制，还没有在实际的组合优化问题中得到运用。

4. 蚁群算法与其他算法的融合

蚁群算法易于与其他算法融合从而互相取长补短，改善算法的性能。目前这个方面的研究成果，包括蚁群算法与遗传算法、神经网络、微粒群算法等之间的融合研究。丁建等利用遗传法生成路径的初始信息素分布，获得了较好的果。对于蚁群算法与遗传算法的融合，研究主要采用遗传算法优化蚁群算法中的参数和信息素，以及将遗传算法中的选择、交叉及变异等操作融入蚁群算法。蚁群算法与神经网络的融合，Blum 等使用蚁群算法训练前馈

神经网络，并将其应用于模式分类。

蚁群算法还可以与免疫算法、PSO 算法等优化算法融合。Holden 等将蚁群算法和 PSO 算法集成在一起，用于处理生物数据集的层次分类（hierarchical classification）。Blo 等提出了基于蚁群算法和粗糙集（rough set）方法的模型，并将其用于特征选择。蚁群算法作为较新的仿生优化方法，与其他算法融合和比较的研究仍处于初级阶段，相究文献和报告较少。因此，设计新的融合策略结合其他算法进一步改善蚁群法的性能，改进蚁群算法与其他算法之间的联系都是非常有意义的研究方向。

3.2.4 蚁群算法的改进研究

1. 精华蚂蚁系统

精华蚂蚁系统（Elitist Ant System，EAS）是对基础 AS 的第一次改进，它在原 AS 信息素更新原则的基础上增加了一个对至今最优路径的强化手段：

$$\begin{cases}\tau(i,j)=(1-\rho)\cdot\tau(i,j)+\sum\limits_{k=1}^{m}\Delta\tau_k(i,j)+e\Delta_b(i,j),\\ \Delta\tau_k(i,j)=\begin{cases}(C_k)^{-1}, & (i,j)\in R^k\\ 0, & \text{其他}\end{cases}\\ \Delta\tau_b(i,j)=\begin{cases}(C_b)^{-1}, & (i,j)\text{ 在路径 }T_{\mathrm{b}}\text{ 上}\\ 0, & \text{其他}\end{cases}\end{cases}\tag{3-10}$$

引入这种额外的信息素强化手段有助于更好地引导蚂蚁搜索的偏向，使算法更快收敛。

2. 基于排列的蚂蚁系统

基于排列的蚂蚁系统（rank-based Ant System，$\mathrm{AS_{rank}}$）在 AS 的基础上给蚂蚁要释放的信息素加上一个权重，进一步加大各边信息素量的差异，以指导搜索。在每一轮所有蚂蚁构建路径后，它们将按照所得路径的长短进行排名，只有生成了至今最优路径的蚂蚁和排名在前（$\omega-1$）的蚂蚁才被允许释放信息素，蚂蚁在边(i,j)上释放的信息素的权重由蚂蚁的排名决定

$$\begin{cases}\tau(i,j)=(1-\rho)\cdot\tau(i,j)+\sum\limits_{k=1}^{\omega-1}(\omega-k)\Delta\tau_k(i,j)+\omega\Delta_b(i,j),\\ \Delta\tau_k(i,j)=\begin{cases}(C_k)^{-1}, & (i,j)\in R^k\\ 0, & \text{其他}\end{cases},\\ \Delta\tau_b(i,j)=\begin{cases}(C_b)^{-1}, & (i,j)\text{ 在路径 }T_{\mathrm{b}}\text{ 上}\\ 0, & \text{其他}\end{cases}\end{cases}\tag{3-11}$$

权重$(\omega-k)$对不同路径的信息素浓度差异起到了一个放大的作用，$\mathrm{AS_{rank}}$ 能更有力度地指导蚂蚁搜索。

3. 最大最小蚂蚁系统

MMAS 在基本 AS 算法的基础上进行了以下改进。

(1) 只允许迭代最优蚂蚁(在本次迭代构建出最短路径的蚂蚁),或者只有最优蚂蚁释放信息素。

(2) 信息素量大小的取值范围被限制在一个区间内。

(3) 信息素初始值为信息素取值区间的上限,并伴随一个较小的信息素蒸发速率。

(4) 每当系统进入停滞状态,问题空间内所有边上的信息素量都会被重新初始化。

4. 蚁群系统

1997 年,蚁群算法的创始人 Dorigo 在文章 *Ant colony system: A cooperative learning approach to the traveling salesman problem* 中提出了一种具有全新机制的 ACO 算法——蚁群系统(Ant Colony System,ACS),进一步提高了 ACO 算法的性能。

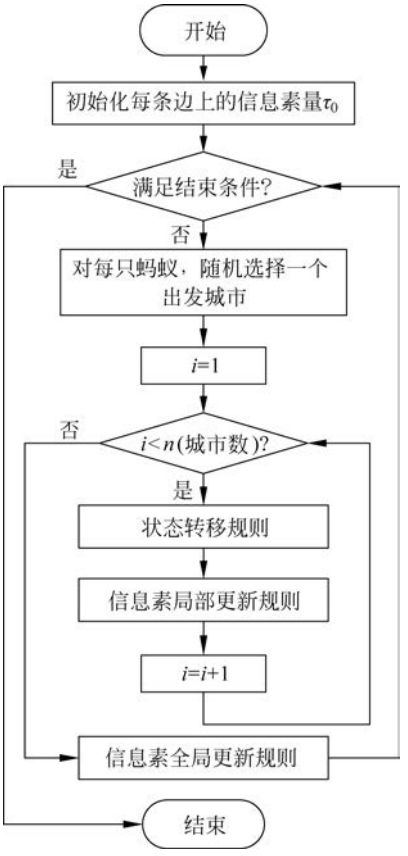


图 3.13 ACS 基本流程

ACS 是 ACO 算法发展史上的里程碑式的作品,其基本流程如图 3.13 所示。

(1) 使用一种伪随机比例规则(pseudo-random proportional)选择下一个城市节点,建立开发当前路径与探索新路径之间的平衡。

$$j=\begin{cases}\arg\max_{j\in J_k(i)}\{[\tau(i,j)],[\eta(i,j)]^{\beta}\}, & q\leqslant q_0\\ S, & \text{其他}\end{cases} \tag{3-12}$$

q_0 是一个[0,1]区间内的参数,当产生的随机数 $q\leqslant q_0$ 时,蚂蚁直接选择使启发式信息与信息素量的指数乘积最大的下城市节点,通常称为开发(exploitation);反之,当产生的随机数 $q>q_0$ 时 ACS 将和各种 AS 算法一样使用轮盘选择策略,我们称为偏向探索(bias exploration)。通过调整 q_0,能有效调节“开发”与“探索”之间的平衡,决定算法是集中开发最优路径附近的区域还是探索其他区域。

(2) 使用信息素全局更新规则,每轮迭代中所有蚂蚁都已构建完路径后,在属于至今最优路径的边上蒸发和释放信息素

$$\begin{cases}\tau(i,j)=(1-\rho)\tau(i,j)+\rho\Delta\tau_b(i,j), & \forall(i,j)\in T_b\\ \Delta\tau_b(i,j)=1/C_b\end{cases} \tag{3-13}$$

不论是信息素的蒸发还是释放,都只在属于至今最优路径的边上进行,这与 AS 算法有

很大的区别。因为 AS 算法将信息素的更新应用到了系统的所有边上，信息素更新的计算复杂度为 $O(n^2)$，而 ACS 算法的信息素更新计算复杂度降低为 $O(n)$。参数 r 代表信息素蒸发的速率，新增加的信息素乘以系数 r 后，更新后的信息素浓度被控制在旧信息素量与新释放的信息素量之间，用一种隐含又简单的方式实现了 MMAS 算法中对信息素量取值范围的限制。

(3) 引入信息素局部更新规则，在路径构建过程中，对每一只蚂蚁，每当其经过一条边 (i,j) 时，它将立刻对这条边进行信息素的更新

$$\tau(i,j)=(1-\rho)\cdot\tau(i,j)+\rho\cdot\tau_0 \tag{3-14}$$

信息素局部更新规则作用于某条边上会使得这条边被其他蚂蚁选中的概率减少。这种机制大大增加了算法的探索能力，后续蚂蚁倾向于探索未被使用过的边，有效地避免了算法进入停滞状态。

图 3.14 描述的是蚁群搜索：顺序构建和并行构建。顺序构建是指当一只蚂蚁完成一轮完整的构建并返回到初始城市之后，下一只蚂蚁才开始构建。并行构建是指所有蚂蚁同时开始构建，每次所有蚂蚁各走一步(从当前城市移动到下一个城市)。对于 ACS 算法，要注意到两种路径构建方式会造成算法行为的区别。在 ACS 算法中通常选择让所有蚂蚁并行地工作。

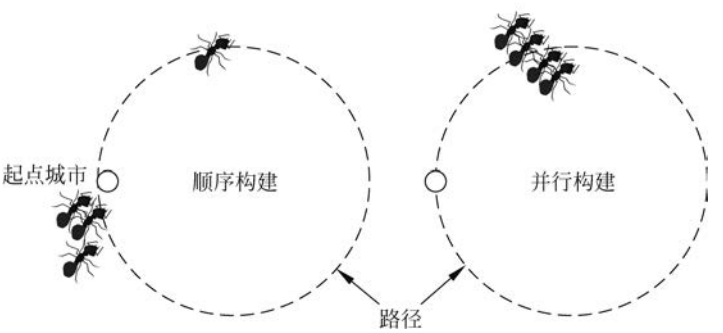


图 3.14　蚁群搜索图

5. 连续正交蚁群系统

近年来，将应用领域扩展到连续空间的蚁群算法也在发展，连续正交蚁群(Continuous Orthogonal Ant Colony，COAC)算法就是其中比较优秀的一种。COAC 算法通过在问题空间内自适应地选择和调整一定数量的区域，并利用蚂蚁在这些区域内进行正交搜索、在区域间进行状态转移、并更新各个区域的信息素，搜索问题空间中的最优解。COAC 的基本思想是利用正交试验的方法将连续空间离散化。

3.2.5　蚁群算法的参数设置

蚁群算法的参数设置如表 3.1 所示。

表 3.1 蚁群算法参数设置表

参　数	参 考 设 置
蚂蚁数目 m	在用 AS 算法、EAS 算法、AS_{rank} 算法和 MMAS 算法求解 TSP 问题时，m 取值等于城市数目 n 时，算法有较好性能；而对于 ACS 算法，$m=10$ 比较合适
信息素权重 α 与启发式信息权重 β	在各类 ACO 算法中设置 $\alpha=1$，$\beta=2\sim5$ 比较合适
信息素挥发因子 r	对于 AS 算法和 EAS 算法，$r=0.5$；对于 AS_{rank} 算法，$r=0.1$；对于 MMAS 算法，$r=0.02$；对于 ACS 算法，$r=0.1$，算法的综合性能较高
初始信息素量 t_0	对于 AS 算法，$t_0=m/C_{nn}$；对于 EAS 算法，$t_0=(e+m)/rC_{nn}$；对于 AS_{rank} 算法，$t_0=0.5r(r-1)/rC_{nn}$；对于 MMAS 算法，$t_0=1/rC_{nn}$；对于 ACS 算法，$t_0=1/nC_{nn}$
释放信息素的蚂蚁个数 w	在 AS_{rank} 算法中，参数 w 设置为 $w=6$
进化停滞判定代数 r_s	在 MMAS 算法中，参数 r_s 设置为 $r_s=25$
信息素局部挥发因子 x	在 ACS 算法中，参数 x 设置为 $x=0.1$
伪随机因子 q_0	在 ACS 算法中，参数 q_0 设置为 $q_0=0.1$

3.2.6 蚁群算法的应用

自从 ACO 算法在一些经典的组合规划问题如 TSP 和 QAP 等 NP-hard 的组合优化问题上取得成功以来，目前已陆续应用到许多新的实际工程领域中。

(1) 在各种工程和工业生产中的应用，例如采用 ACO 算法的思想求解大规模集成电路综合布线问题。在布线过程中，各个引脚对蚂蚁的引力可根据引力函数来计算。各个线网智能体根据启发策略，像蚁群一样在开关盒网格上爬行，所经之处便布上一条金属线，历经一个线网的所有引脚之后，线网便布通了。

(2) ACO 算法在各种实际规划问题中的应用，例如在机器人路径规划中的应用。机器人作为一种智能体，在复杂工作环境下的路径规划问题、多机器人之间的协作策略问题，在很大程度上类似于蚂蚁觅食优选路径以及蚂蚁群体中个体之间通过信息素形成协作。路径规划算法是实现机器人控制和导航的基础之一，实验证明利用 ACO 算法解决该问题有很大的优越性。

另外，ACO 算法在动态优化组合问题中也有应用，具体是在有向连接的网络路由和无连接网络系统路由中的应用。其他应用还包括蚂蚁人工神经网络、车辆路线问题(Vehicle Routine Problem，VRP)以及在图像处理和模式识别领域的应用等。

1. 静态组合优化问题

(1) 典型的组合优化问题。从最初用蚁群算法解决旅行商问题开始，研究者陆续将其应用到其他典型的组合优化问题：二次规划问题(quadratic assignment problems)、图着色问题 (graph coloring problems)。这些问题具有很强的工程代表性，蚁群算法在典型的组

合优化问题上的出色表现加速了它在工程应用领域的发展。

(2) 物流领域的应用物流领域中的一些问题也具有组合优化问题的性究较多的是物流配送领域的车辆路径问题,Gambardell 最先将蚁群算法应用于车辆路径问题。Merle 等将蚁群算法应用于资源项目约束的排定问题(resource constrained project scheduling problems),实验表明与其他启发式法相比较优势明显。

(3) 机构同构判定问题。在机械设计领域普遍存在的机构同构判定问题,将该类问题转化为求其邻近矩阵的特征编码值的问题,利用蚁群的强大的搜索能力进行求解,在参数选择合适的情况下,可以取得令人满意的结果。

(4) 电力系统领域的应用。电力系统的许多优化问题本质上属于组合优化问题。Gomez 等将蚁群算法应用于配电网络的规划。Viacho Giannini 提出了用蚁群算法解决约束潮流问题,实验结果表明蚁群算法具有较高的可靠性和优化能力。Fong 等将蚁群算法应用到发电厂检修计划的优化。

2. 动态组合优化问题

在动态优化组合问题中,可以分为有向连接的网络路由和无连接网络系统路由。

(1) 有向连接的网络路由。在有向连接的网络中,同一个话路的所有数据包沿着一条共同路径传输,这条路径由一个初步设置状态选出。Schoonderwerd 等首先将蚁群算法应用于路由问题,后来 White 等将 ACO 算法用于单对单点和单对多点的有向连接网络中的路由,Bonabeau 等通过引入动态规则机制改善蚁群算法,Dorigo 研究将蚁群算法用于高速有向连接网络系统中,得到公平分配效果最好的路由。

(2) 无连接网络系统路由。随着 Internet 规模不断扩大,在网络上导入 QoS 技术以确保实时业务的通信质量。QoS 组播路由的目的是在分布的网络中寻找最优路径,要求从源节点出发,历经所有的目的点节点,并且在满足所有约束条件下,达到花费最小的服务水平。应用蚁群研究解决包含带宽、延时、延时抖动、包丢失率和最小花费约束等约束条件在内的 QoS 组播路由问题,效果优于模拟退火算法和遗传算法。

3. 其他应用领域

蚁群优化算法的其他应用还包括学习模糊规则问题、蚂蚁自动规划设计、蚂蚁人工神经网络等。

3.2.7 蚁群算法的相关应用与 MATLAB 算例

【例 3-5】 已给一个 n 个点的完全图,每条边都有一个长度,求总长度最短的经过每个顶点正好一次的封闭回路。

解:

(1) **蚁群算法原理**。蚂蚁在路径上释放信息素,当蚂蚁碰到还没走过的路口,就随机挑选一条路走,同时释放与路径长度有关的信息素,信息素的浓度与路径长度成反比。后来的

蚂蚁再次碰到该路口时，蚂蚁在选择下一个要转移的城市时候是基于概率选择的，选择信息素浓度较高的路径，最优路径上的信息素浓度越来越高，最终蚁群找到最优寻食路径。

(2) **蚁群算法与 TSP 问题**。将 m 个蚂蚁随机地放在多个城市，让这些蚂蚁从所在的城市出发，n 步(一个蚂蚁从一个城市到另外一个城市为一步)之后返回出发的城市。如果 m 个蚂蚁走出的 m 条路径对应的最短者不是 TSP 问题的最短路程，则重复这一过程，直至寻找到满意的 TSP 问题的最短路径为止。

(3) **路径构建**。蚁群系统中的随机比例规则：对于每只蚂蚁，路径记忆向量按照访问顺序记录了所有已经经过的城市序号。设蚂蚁 k 当前所在城市为 i，其选择城市作为下一个访问对象的概率为

$$p_k(i,j)=\begin{cases}\dfrac{[\tau(i,u)]^{\alpha}[\eta(i,j)]^{\beta}}{\sum\limits_{j\in J_k(i)}[\tau(i,u)]^{\alpha}[\eta(i,u)]^{\beta}}, & j\in J_k(i)\\ 0, & \text{其他}\end{cases} \tag{3-15}$$

其中，在 TSP 问题中，每次循环当中，每只蚂蚁走出的每条路径为 TSP 问题的候选解，m 只蚂蚁一次循环走出的 m 条路经为 TSP 问题的一个解子集，所以这个解子集越大则算法的全局搜索能力越强，但是过大会使算法的收敛速度降低。如果 m 太小，算法也很容易陷入局部最优，过早地出现停滞现象。α 为信息素重要程度因子，反映了蚂蚁在从城市 i 向城市 j 移动时，这两个城市之间道路上累积的信息素在指导蚂蚁选择城市 j 的程度，即蚁群在路径搜索中随机性因素作用的强度。α 值越大，蚂蚁选择之前走过的路径的可能性越大，搜索路径的随机性减弱，α 越小，蚁群搜索范围就会减小，容易陷入局解最优。β 为启发函数重要程度因子，β 值越大，蚁群就越容易选择局部较短路径，这时算法的收敛速度是加快了，但是随机性却不高，容易得到局部的相对最优。τ 为禁忌表，用于存放第 k 只蚂蚁已经走过的城市。根据式(3-14)，τ 与 ρ 密切相关，ρ 为信息素挥发因子，且 $0\leqslant\rho\leqslant1$，$1-\rho$ 为信息残留因子。ρ 过小时，在各路径上残留的信息素过多，导致无效的路径继续被搜索，影响到算法的收敛速率；ρ 过大，虽然可以排除搜索无效的路径，但是不能保证有效的路径不会被放弃搜索，影响到最优值的搜索，在 AS 算法中通常设置为 $\rho=0.5$。

经过 150 次迭代，对 50 个城市求最优路径，结果是 24722.0443，执行后的效果如图 3.15 和图 3.16 所示。

3.2.8 蚁群算法的总结与展望

蚁群算法作为一种新的仿生启发式优化算法，虽然刚问世不久，但它在解决复杂组合优化问题方面，显示出了明显的优势。蚁群算法具有较强的鲁棒性、通用性和并行搜索等优点，但搜索时间较长，在算法模型、收敛性及理论依据等方面还有许多工作有待进一步深入研究。蚁群算法在如下几个方面可以做进一步的探讨。

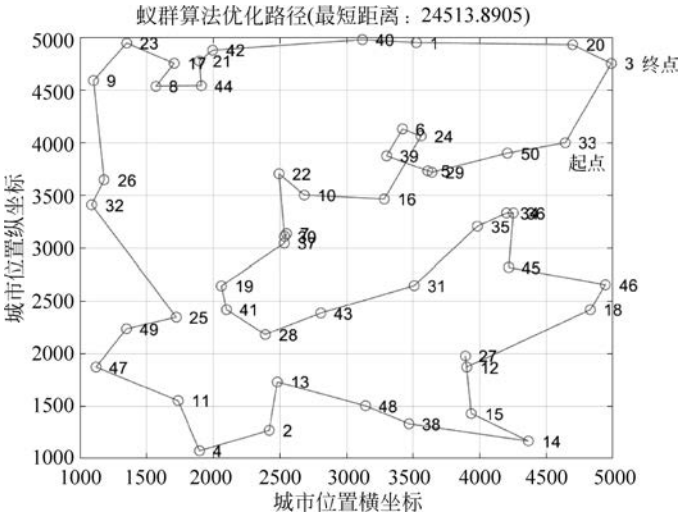


图 3.15　蚁群算法优化路径

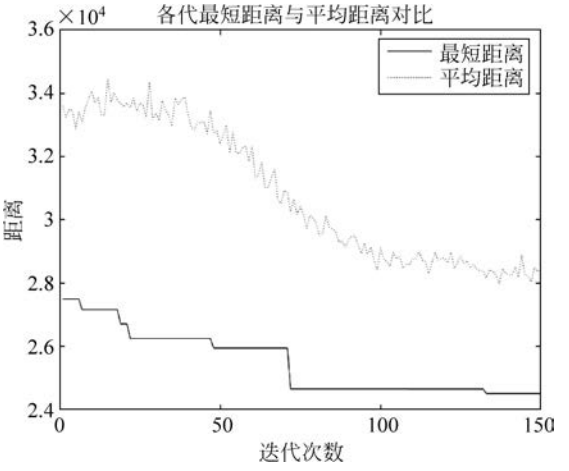


图 3.16　蚁群算法效果

1. 蚁群算法基础数学理论的研究

蚁群算法的发展需要坚实的理论基础，目前这方面的研究成果还比较匮乏。虽然可以证明某几类蚁群的收敛性，目前收敛性的证明并没有说明要找到至少一次最优解需要的计算时间，即使算法能够找到最优解，付出的计算时间也可能是个天文数字。此外，蚁群算法收敛的严格数学证明，在更强的概率意义下的收敛条件，蚁群算法中信息素挥发对算法收敛性的影响，蚁群算法动力模型及根据其动力学模型对算法进行性能分析，蚁群算法最终收敛至全局最优解时间的复杂度等问题也需要进一步的研究。运用蚁群算法处理各种问题时，选择什么样的编码方案、什么样的参数组合以及如何设置算法中人工信息素等，只能具体问题具体分析，目前并没有通用的、严密的、科学的模型和方法。要想进一步推动蚁群算法的

应用和发展，就迫切需要宏观理论的指导。

2. 蚁群算法自身的发展

可以考虑从新的角度对蚁群算法进行改进，比如概率方法、多种群策略、蚁群之间信息共享机制的改进以及引入其他优化算法等，蚁群算法是基于种群的方法(population based method)，具有并行性，可以进一步对算法的并行化进行研究。在蚁群算法自身方面加大改进，控制参数范围，避免由于参数过大或者过小而陷入局部最优。

3. 与其他算法的比较与结合

蚁群算法应用领域的拓宽还应与其他相关学科进行交叉研究。蚁群算法目前最为成功的应用是大规模的组合优化问题，下一步应将蚁群引入到更多的应用领域(如自动控制和机器学习等)，并与这些相关的学科进行深层次的交叉研究，进一步促进算法的研究和发展。此外，蚁群具有很强的耦合性，易与其他传统优化算法或者启发式算法结合，但 M. Dorigo 博士指出，蚁群算法与分布估计算法(Estimation of Distribution Algorithm，EDA)、图模型(graphical model)和贝叶斯网络(Bayesian network)等概率学方法之间的关系尚不明确，这方面的工作还需要继续探索下去。以后研究中应以耦合算法为一个重要研究方向，将蚁群和其他仿生算法结合，以达到取长补短的效果。近期已经取得一定成果的是与免疫算法的结合以及和遗传算法的结合，和其他算法的融合有待进一步的扩展。

4. 蚁群算法应用领域的拓展

蚁群算法已经被引入很多领域发挥其优化能力。目前应用较多的是静态组合优化问题，如何改进将其应用于动态组合优化问题和连续优化问题是一个值得探索的方向。